Date: December 2012

OMG hData RESTful Transport

OMG Document Number: dtc/12-12-20
Standard document URL: http://www.omg.org/spec/hdata/1.0
Associated Schema Files: none

This OMG document replaces the submission document (health/2011-09-04, alpha). It is an OMG Adopted Beta specification and is currently in the finalization phase. Comments on the content of this document are welcome, and should be directed to issues@omg.org.

You may view the pending issues for this specification from the OMG revision issues web page: http://www.omg.org/issues/.

The FTF Recommendation and Report for this specification will be published on September 21, 2012. If you are reading this after that date, please download the available specification from the OMG Specifications Catalog.

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION,

INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management GroupTM, OMGTM , Unified Modeling LanguageTM, Model Driven Architecture LogoTM, Model Driven Architecture DiagramTM, CORBA logosTM, XMI LogoTM, CWMTM, CWM LogoTM, IIOPTM , IMMTM, MOFTM , OMG Interface Definition Language (IDL)TM , and OMG Systems Modeling Language (OMG SysML)TM are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents, Report a Bug/Issue (http://www.omg.org/technology/agreement.htm).

Table of Contents

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling LanguageTM); CORBA® (Common Object Request Broker Architecture); CWMTM (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org/.*

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A catalog of all OMG Specifications is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM)

Platform Specific Model and Interface Specifications

- CORBAservices

- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications.

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA

Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

1 Scope

The hData RESTful application programming interface (API) specification defines remote operations for accessing components of a Health Record and sending messages to an EHR system. "RESTful" refers to a style of web services in which resources are identified by URLs and clients use stateless HTTP operations to perform operations on those resources [14].A related specification, the HL7 hData Record Format (HRF) [1], describes the logical organization of the information in an electronic health record (EHR). Please refer to the HRF specification for more details on the HRF and how it fits into the HL7 version 3 standards.

As described in more detail in section 9 of this specification, the hData specification is a platform specific module (PSM) for the OMG Retrieve, Locate, Update Service (RLUS) platform independent model (PIM) (see [13]). It implements the RLUS PIM Management and Query Interface using a RESTful architectural style. The current specification is specific to the current RLUS PIM. If RLUS evolves and adds new functionalities and conformance profiles, the hData specification will need to be updated to reflect these changes accordingly.

In addition to the RLUS conformant operations, this specification introduces a number of additional features and operations specific to the operational needs of REST compliant set of HTTP services in the context of health information exchange: section 7 describes a mechanism to provide receipt confirmation of the message exchange to both client and server. Section 8 provides a general framework for the server to provide information about the supported security mechanisms to the client.

2 Namespaces

This document uses the following namespaces, which are originally defined in the HL7 HRF specification [1]. This specification uses a number of namespace prefixes throughout, as listed in Table 1. Note that the choice of namespace prefix is arbitrary and not semantically significant.

Namespace Prefix	Namespace URI	Description
hrf	http://www.hl7.org/schema/hdata/2009/06/core	Namespace for elements in this document
hrf-md	http://www.hl7.org/schema/hdata/2009/11/meta	SectionDocument metadata

3 Glossary (non-normative)

HL7 hData Record Format (HRF) – a related specification that specifies an abstract hierarchical organization, packaging, and metadata for individual documents (referred to as "Section Documents" within the HRF specification). Section Documents can be of any type, either XML documents (such as CDA documents, H7v3 messages, or simplified XML wire formats, etc.) or of other media types (such as e.g. MS Word documents or DICOM files). Also contained in this specification is the format for specifying the content that goes into an hData record, which is called the hData Content Profile (HCP) format.

hData Record (HDR) - an single instantiation of the HRF.

OMG hData Restful Transport – the current specification, defining how the abstract hierarchical organization defined

within the HRF specification is accessed and modified through a RESTful approach, using HTTP as the access protocol. It creates a unique mapping to an URL structure, and defines how HTTP verbs such as GET, PUT, DELETE, etc. affect the underlying information.

hData Content Profile (HCP) - a profile of the content of an HDR. The HRF specification contains the definition of the HCP format.

RLUS – a Retrieve, Location, and Update Service, as defined jointly by OMG and HL7.

Semantic Signifier - a structure definition (such as a schema) and an associated set of validation instructions. The semantic signifier describes the structural and semantic definition of the logical records managed by RLUS.

4 Conformance Point

Implementations must support the entire mapping of this specification excluding non-normative portions. The keywords "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

5 Additional Information

5.1 Acknowledgements

The following contributed to this publication:

- Nick Dikan
- Robert Dingwell
- Andrew Gregorowicz
- Grahame Grieve
- Marc Hadley
- Paul Knapp
- Mark Kramer
- John Koisch
- Stefano Lotti
- Anil Luthra
- Galen Mulrooney
- Dale Nelson
- Ken Rubin
- Samuel Sayer
- Harry Sleeper
- Andy Stechishin

Editor: Gerald Beuchelt

6 hData Record RESTful Transport

6.1 Overview

Any instantiation of an HRF – called an hData Record (HDR) – can be represented as a set of Hypertext Transfer Protocol (HTTP 1.1, see [8]) resources in a canonical way by mapping the hierarchical structure of the HDR to a URL resource hierarchy underneath the *base URL* (see below). Each HDR Section and Section Document is represented by a unique URL, which is constructed from the Section paths and Section Document names. The entire HDR is referenced by a base URL that depends on the implementation. See IETF RFC 3986, section 5 for more details. This base URL will be denoted as *base URL* throughout this document.

6.1.1 Out of Scope

While this specification does not dictate the format of the *baseURL*, the *base URL* MUST NOT contain a query component. All content within an HDR that uses this transport specification MUST be expressible as a HTTP resource. In the following, the minimum version for HTTP is 1.1.

This specification does not address data modeling in any form. hData is designed to be able to transport clinical data of any Internet Media Type. The HL7 HRF specification describes how established and emerging data models can be used through the hData Content Profile mechanism by hData-enabled systems.

It should be noted that this specification was designed with extensibility in mind, e.g. by not defining certain HTTP methods on classes of HTTP resources. When implementers use these extension points, the interoperability assertion of this specification does not extend to such extensions, but only covers those parts of an implementation that are in conformance with this document. At the same time, implementers MUST implement all mandatory elements of this specification.

6.1.2 General Conventions and Considerations

Default Response Codes

Any HTTP GET, PUT, POST, DELETE, or OPTIONS operation (see [8], section 9) on a given resource that is not implemented MUST return an HTTP response with a status code of 405 that includes an Allow header that specifies the allowed methods. All operations SHOULD return HTTP status codes in the 5xx range if there is a server problem. Other HTTP status codes MAY be added by security mechanisms or other extensions.

Incorrect Updates

Servers are permitted to reject update operations because of integrity concerns or business rules implemented on the server, and return HTTP status code 409 Conflict. .

Recommended HTTP Headers

It is RECOMMENDED that all section document responses include a “Last-Modified” header. It is RECOMMENDED that all document resources support the “If-ModifiedSince” and “If-Unmodified-Since” headers to support conditional GET and optimistic concurrency.

Content Compression

For improved performance it is RECOMMENDED that the server support client requests for GZIP compression. Clients will request compression by setting the Accept-Encoding HTTP header to “gzip.” The server SHOULD honor this request for all documents, so that devices may benefit from the reduced bandwidth needs and improved battery life when

requesting compressed content.

Content Negotiation

All URLs within this section MUST support HTTP content negotiation through the use of the HTTP Accept header (see [8]). The client MUST NOT ask for media types not supported by the Content Model. If the client requests an unsupported media type, the server MUST return a HTTP status code 415.

In addition, the service SHOULD support alternate content negotiation by allowing to append a HTTP query string of the following form: (*URL*)?$format=(*mediaType*) where (URL) is the URL for the resource and (mediaType) is a valid media type supported by the content type of the resource. Since XML and JSON based formats will be commonly used, clients MAY use "xml" instead of "text/xml" and "json" instead of "application/json", respectively. Servers MUST understand these abbrevated media types.

For Section-level URLs and query responses it is RECOMMENDED to allow representation of the Atom feed in at least the standard application/atom+xml media type, as well as in application/json media type. This is achieved by representing the Atom feed through a JSON object. This object is built as follows:

- The object SHOULD have an attribute "updated" with the time of the completion of the query rendered by calling the JavaScript Date.toString method.
- The object SHOULD include an attribute "self" with a URL pointing to the URL corresponding to the query.
- The object MUST include an array called "entries" that contains objects corresponding to the results of the query.

The objects contained in the entries correspond to SectionDocuments, and each object in the array MUST contain the following attributes:

- An attribute called "id" that is set to the path segment of the *documentname* in the URL for the SectionDocument (see section 6.5).
- An attribute called "self" that MUST contain a URL which points to the (*resourceURL*), as defined in section 6.5.
- An attribute called "updated" that MUST contain the time of the last update to the SectionDocument. It is rendered according to the rules of ECMAScript Date interchange format (see ECMA-262, section 15.9.1.15).

Example (non-normative):

```
{
  "updated":"2012-10-21T12:34:28Z",
  "self":http://example.com/foo/bar",
  "entries":[
    {
      "id":"123456",
      "self":http://example.com/foo/bar/123456",
      "updated":"2012-10-21T12:34:28Z"
    },
    {
      "id":"9876",
      "self":http://example.com/foo/bar/9876",
```

```
      "updated":"2012-10-21T12:34:28Z"
    }
  ]
}
```

This example is equivalent to the following Atom feed:

```
<?xml version="1.0" encoding="utf-8"?>
 <feed xmlns="http://www.w3.org/2005/Atom">
   <title>Example Query</title>
   <link rel="self"
        href="http://example.com/foo/bar"/>
   <updated>2012-10-21T12:34:28Z</updated>
   <entry>
     <title>123456</title>
     <link rel="self" href="http://example.com/foo/bar/123456"/>
     <id>123456</id>
     <updated>2012-10-21T12:34:28Z</updated>
   </entry>
  <entry>
     <title>9876</title>
     <link rel="self" href="http://example.com/foo/bar/9876"/>
     <id>9876</id>
     <updated>2012-10-21T12:34:28Z</updated>
   </entry>
 </feed>
```

Forbidden Keywords

To allow optional extensions, the *sectionpath* and the SectionDocument *documentname* MUST NOT use any of the following keywords

- history
- root
- search
- validate

Intermediaries Consideration

The HTTP protocol may be routed through an HTTP proxy such as squid. Such proxies are transparent to the applications, though implementors should be alert to the effects of rogue caching. Interface engines may also be placed between the consumer and the provider; these differ from proxies because they actively alter the content or destination of the HTTP exchange, and are not bound by the rules that apply to HTTP proxies. Such agents are allowed, but MUST mark the HTTP header to assist with troubleshooting. Any agent that modifies an HTTP request or response content other than permitted under the rules for HTTP proxies must add a stamp to the HTTP headers like this:

- request-modified-[identity]: [purpose]
- response-modified-[identity]: [purpose]

The identity must be a single token defined by the administrator of the agent that will sufficiently identify the agent in the context of use. The header must specify the agent's purpose in modifying the content. End point systems must not use this header for any purpose; its aim is to assist with system troubleshooting.

6.2 Operations on the Base URL

6.2.1 GET

If there is no HDR at the base URL, the server SHOULD return a 404 - Not found status code.
The server MUST offer an Atom 1.0 compliant feed of all child sections specified in the HRF specification [1], as identified in the corresponding sections node in the root document.

It is RECOMMENDED that the server also offers a web user interface that allows users to access and manipulate the content of the HDR, as permitted by the policies of the system. Selecting between the Atom feed and the user interface can be achieved using standard content negotiation (HTTP Accept header). This is not necessary for systems that are used by non-person entities only. If the Accept header is non-existent, or set to */* or application/atom+xml, the system MUST return the Atom feed. For all other cases the format of the returned resource is left to the implementer.

Status Code: 200, 404

6.2.2 POST – Parameters:extensionId, path, name

This operation is used to create a new Section at the root of the document. The request body is of type "application/x-www-form-urlencoded" and MUST contain the extensionId, path, and name parameters. The extensionId parameter MAY be a string that is equal to value of one of the registered <extension> nodes of the root document of the HDR identified by *base URL*. The path MUST be a string that can be used as a URL path segment. If any parameters are incorrect or not existent, the server MUST return a status code of 400.

The system MUST confirm that there is no other section registered as a child node that uses the same path name. If there is a collision, the server MUST return a status code of 409.

If the extensionId is not registered as a valid extension, the server MUST verify that it can support this extension. If it cannot support the extension it MUST return a status code of 406. It MAY provide additional entity information. If it can support that extension, it MUST register it with the root document of this record.

When creating the section resource, the server MUST update the root document: in the node of the parent section a new child node must be inserted. If successful, the server MUST return a 201 status code and SHOULD include the location of the new section. The name parameter MUST be used as the user-friendly name for the new section.

Status Code: 201, 400, 406, 409

6.2.3 PUT

This operation is undefined by this specification.

Status Code: 405, unless an implementer defines this operation.

6.2.4 DELETE

This operation is undefined by this specification.

Status Code: 405, unless an implementer defines this operation.

6.2.5 OPTIONS

The OPTIONS operation on the *base URL* is per [8], section 9.2, intended to return communications options to the clients. Within the context of this specification, OPTIONS is used to indicate which security mechanisms are available for a given *base URL* and a list of hData content profiles supported by this implementation. All implementations SHOULD support OPTIONS on the *base URL* of each HDR and return a status code of 200, along with:

- The WWW-Authenticate HTTP header defined in RFC 2617 [6]. The security mechanisms defined at the *baseURL* are applicable to all child resources, i.e., to the entire HDR. Refer to section 8 for a detailed discussion on the semantics of this field.
- An X-hdata-hcp HTTP header that contains a space separated list of the identifiers of the hData Content Profiles supported by this implementation.
- The X-hdata-extensions HTTP header contains a space separated list of the identifiers of the hData extensions supported by this implementation independent of their presence in the root document at *base URL*/root (cf. section XXX in [1] describing the root document format for an explanation of the extensions in a root).

The server MAY include additional HTTP headers. The response SHOULD return an HTTP body with the document identified in section 6.3.2. The client MUST NOT use the Max-Forwards header when requesting the security mechanisms for a given HDR. If it does, the server MUST return a 403 Forbidden status code with optional message "Request cannot include Max-Forwards header field".

If the baseURL does not correspond to an HDR, it should not respond with a successful OPTIONS response, but return a 404 error.

Status Code: 200, 403, 404

6.3 Special Paths on *baseURL*

6.3.1 *baseURL*/root

6.3.1.1 GET

This operation MUST return an XML representation of the current root document, as defined by the HRF

specification. If the server supports non-XML representations of the root Document, the client MAY request

different media types through the content negotiation mechanisms.

If there is no HDR at the base URL, the server SHOULD return a 404 - Not found status code.

Status Code: 200, 404, 415

6.3.1.2 POST, PUT, DELETE

These operations MUST NOT be implemented.

Status Code: 405

6.3.2 *baseURL*/metadata

6.3.2.1 GET

The resource at this URL represents the metadata associate with the operations of this service. Any request to this URL MUST be completed without prior authentication or authorization. The service MUST return an XML document describing implementation specific metadata. The semantics of this records are identical to the values of the header fields in 6.2.5.

Status Code: 200, 415

6.3.2.2 POST, PUT, DELETE

These operations MUST NOT be implemented.

Status Code: 405

6.3.3 *baseURL*/search

6.3.3.1 GET

Please refer to section 6.6 for a definition of the query mechanism.

6.3.3.2 POST, PUT, DELETE

These operations MUST NOT be implemented.

Status Code: 405

6.4 *baseURL*/*sectionpath*

6.4.1 GET

This operation MUST return an Atom 1.0 [3] compliant feed of all section documents and child sections contained in this section. Each entry MUST contain a link to a resource that uniquely identifies the section document or child section. For section documents, this link MUST be the version specific link (see section 6.5). If the section document type defines a creation time, it is RECOMMENDED to set the Created node to that datetime. For documents that were deleted using the DELETE operation in section 6.5.4, the server SHOULD create an Atom entry with a <at:deleted-entry> node set as described in section 6.5.4.

For section documents, the Atom Content element MUST contain the XML representation of its metadata (see [1], Section 2.4.1).

If baseURL does not exist or no sectionpath of name sectionpath exists, the implementation MUST return a HTTP status code 404.

Status Code: 200, 404, 415

6.4.2 POST

For creating a new sub section, three additional parameters are used, and the POST will create a new child section within this section. For new documents a document MUST be sent that conforms to the business rules expressed by the extension that the section has registered.

6.4.2.1 Add new section – Parameters: extensionId, path, name

The content type MUST equal "application/x-www-form-urlencoded" for the POST method to create a new sub section. The extensionId parameter is the URI in the root document that identifies the Extension element. If the extensionId is not registered as a valid extension, the server MUST verify that it can support this extension. If it cannot support the extension it MUST return a status code of 406 and MAY provide additional information in the entity body. If it can support that extension, it MUST register it with the root of this record. The path MUST be a string that can be used as a URL path segment. The name parameter MUST be used as the user-friendly name for the new section. If any parameters are incorrect, the server MUST return a status code of 400.

The system MUST confirm that there is no other section registered as a child node that uses the same path name and that it can create a new subsection identified by the path parameter. If there is a collision, the server MUST return a status code of 409.

When creating the section resource, the server MUST update the root document: in the node of the parent section a new child node must be inserted. When creating a sub-section resource, the server MUST also update the Atom 1.0 [3] compliant feed for the parent section: a new child node must be inserted as an entry in the Atom content with a link to the new section. The server MUST return a 201 status code. The extensionId and path parameters are REQUIRED, the name parameter is OPTIONAL.

Status Code: 201, 400, 406, 409

6.4.2.2 Add new document

When adding a new section document, the request Content Type MUST be "multipart/form-data" if including metadata. In this case, the content part MUST contain the section document. The content part MUST include a Content-Disposition header with a disposition of "form-data" and a name of "content." The metadata part MUST contain the metadata for this section document. The metadata part MUST include a Content-Disposition header with a disposition of "form-data" and a name of "metadata." It is to be treated as informational, since the service MUST compute the valid new metadata based on the requirements found in the HRF specification. The content media type MUST conform to the media type of either the section or the media type identified by metadata of the section document. For XML media types, the document MUST also conform to the XML schema identified by the extensionId for the section or the document metadata. If the content cannot be validated against the media type and the XML schema identified by the content type of this section, the server MUST return a status code of 400.

If POST does not include metadata then MUST POST with a Content Type conforming to the media type of the section. The body of the POST MUST contain the document of the new document. Document metadata in this case MUST be created by the system, based on instructions in the hData Content Profile applying to the system.

The server MAY support bulk updates of sections: if the media type is set to "application/xml+atom", the client MUST send a valid Atom 1.0 feed that complies with the requirements for the Atom feed provided by the section. The server will parse the feed and add any <atom:entry> as a new SectionDocument to the section. If the client intends to update a server resource, it MUST include the server URL for that resource in the <atom:link rel="self"> element of the entry. If the client intends to create a new SectionDocument, it MUST NOT use <atom:link rel="self">.

If the POSTed feed contains entries that the server already knows about, it MUST update these entries if they are different from the server's version.

If the request is successful, the new section document MUST show up in the document feed for the section. The server returns a 201 with a Location header containing the URI of the new document.

Status Code: 201, 400

6.4.3 PUT

This operation is not defined by this specification.

Status Code: 405, unless an implementer defines this operation.

6.4.4 DELETE

This operation MAY be implemented, but special precaution should be taken: if a DELETE is sent to the section URL, the **entire** section, its documents, and subsections are completely deleted. Future requests to the section URL MUST return a status code of 404, unless the record is restored. If successful the server MUST return a status code of 204. If DELETE is implemented, special precautions should be taken to assure against accidental or malicious deletion. Future requests to the section URL MAY return a status code of 410, unless the record is restored.

Status Code: 204, 404, 410

6.5 *baseURL/sectionpath/documentname*

The resource URL for a SectionDocument is constructed as follows:

(*resourceURL*) = *baseURL*/*sectionpath*/*documentname*

Note that the sectionpath may contain more than one path segment. To support the functionality described in this section, the (resourceURL) can be extended in different ways.

Versioning

To support versioning and optimistic concurrency control (OCC) of SectionDocuments this specification uses a version-aware URL naming scheme. The implementation MUST support the following extension mechanism:

(*versionAwareResourceURL*) = (*resourceURL*)/history/(*versionId*)

where *versionId* can be any permissible URL-encoded string. The server MUST support version specific retrieval of the current version, and MAY support version specific retrieval of older versions.

Note that other HTTP-based protocols use ETags for caching and OCC. This specification does not use ETags for two reasons: (i) ETags are optional in HTTP 1.1 and especially disadvantaged devices may decide not to support them. (ii) ETags have been used in the past to enable privacy violating tracking. As a result, some user may decide to disable ETags altogether.

Validation

Similar to the version-aware resource structure, the following URL for sectionDocuments is reserved:

(*validationResourceURL*) = (*resourceURL*)/validate/

This allows a client to ask the server whether the attached POSTed content would be acceptable as a PUT. By this means that client can

- validate business logic during testing concerning correct content, and
- implement a light-weight two-phase commit alternative that reduces the chance of partial success of multi-resource operations.

Implementation of the validation mechanism is NOT REQUIRED.

6.5.1 GET

A GET on the (*resourceURL*) MUST return the representation of the document that is identified by *documentname* within the section identified by *sectionpath* in the HTTP body, a status code of 200, and a Content-Location header containing the (*versionAwareResourceURL*) of the current version of that resource. The *documentname* is typically assigned by the underlying system and is not guaranteed to be identical across two different systems. Implementations MAY use identifiers contained within the infoset of the document as *documentnames*. If the resource was deleted with a DELETE, the service SHOULD return a HTTP status code 410, with no body. It MAY include a (*versionAwareResourceURL*) that points to the last valid version before it got deleted. Alternatively it MAY return a 404 status code instead.

If no document of name *documentname* exists, the implementation MUST return a HTTP status code 404.

Status Codes: 200, 404, 410, 415

6.5.2 POST

This operation is used to replace metadata on a section document. When replacing the metadata, the hrf-md:DocumentId MUST NOT be changed – the server MUST return a status code 403 if this is attempted. This operation SHOULD NOT be used unless necessary for replicating information within an organization. If a section document is copied from one system to another, a new document metadata instance MUST be constructed from the original metadata according to the rules in the HRF specification.

The request Media Type MUST be application/xml. The body MUST contain the document metadata. It MUST conform to the XML schema for the document metadata, defined in [1]. If the metadata is not of media type application/xml or it cannot be validated against the document metadata XML schema, the server MUST return a status code of 400.

If the request is successful, the document metadata for the section document MUST be updated. The server returns a 201.

Status Code: 201, 400, 403

6.5.3 PUT

This operation is used to update a document by replacing it. The content MUST conform to the media type identified by the document metadata or the section content type. For media type application/xml, the document MUST also conform to the XML schema that corresponds to the content type identified by the document metadata or the section. If the parameter is incorrect or the content cannot be validated against the correct media type or the XML schema identified by the content type of this section, the server MUST return a status code of 400.

To enable safe updates, the following process MUST be used:

- The client reads obtains the representation of the current version by performing a GET on the (*resourceURL*). This contains the reference to (*versionAwareResourceURL*).
- The client makes the necessary changes to the state.
- The client MUST then PUT the updates representation to the (*resourceURL*) and quote the (*versionAwareResourceURL*) in the Content-Location header of the PUT operation.
- If the (*versionAwareResourceURL*) provided by the client is the current version, the server accepts the representation and persists the change, and returns a HTTP response with a status code of 202 OK, a Content-

Location header with the (*versionAwareResourceURL*) of the new version, and the representation of the new version of the resource in the response

- If the (*versionAwareResourceURL*) no longer represents the current version, the server MUST return a HTTP response with status code 412, a Content-Location header with the new (*versionAwareResourceURL*), and a representation of the latest version of the resource.

If the request is successful, the updated section document MUST show up in the document feed for the section. The server returns a 200.

This operation MAY be used to create a new document at a specific location, if the client desires to suggest naming of the resource. The document creation rules in section 6.4.2.2 apply to this operation with the following changes:

- If the server cannot create a document this way, it MUST return a HTTP status code of 409 – Conflict.
- Bulk updates MUST NOT be attempted through this operation. If the client sends an Atom media type, the server MUST return a status code of 415 – Unsupported Media type.

Status Code: 200, 400, 409, 412, 415

6.5.4 DELETE

This operation MAY be implemented. If a DELETE is sent to the document URL, the document is completely deleted. If DELETE is implemented, special precautions should be taken to assure against accidental or malicious deletion. Future requests (GET, PUT, POST, DELETE) to the document URL MAY return a status code of 410, unless the record is restored. In this case, the Atom feed for the Section formerly containing the SectionDocument SHOULD inject a <at:deleted-entry> node into the feed as described in [12] and use the same representation in the JSON representation, if used. If baseURL, sectionpath, or target document name does NOT exist, the implementation SHOULD return a 404 - Not found HTTP status code.

Status Code: 204, 404, 410

6.6 Queries

To support simple queries over the entire hData Record as identified by the (*baseURL*) as well as individual Sections identified by (*baseURL*)/(*sectionpath*) the system MUST support the following extension:

(*baseURL*)/search?(*querystring*)

This allows a search over the entire record. The (*querystring*) can be arbitrary, but it is RECOMMENDED to use HTML form-encoding style key/value parameters, as documented in HTML 4.01, section 17.13.4. The server returns the results of the query in the form of an Atom feed, with each Entry representing a search result.

(*baseURL*)/(*sectionpath*)/search?(*querystring*)

The semantics for this query are identical to the one above., but the scope is limited to the section described by the (*sectionpath*).

The entries in the Atom feed (or its JSON representation) have a defined order, though there is not necessarily an underlying meaning in the ordering. Other specifications using the hData REST Transport MAY assign a specific semantic to the ordering of the Atom feed resulting from a query."

7 Complex Operations

7.1 Reliable Operation Pattern

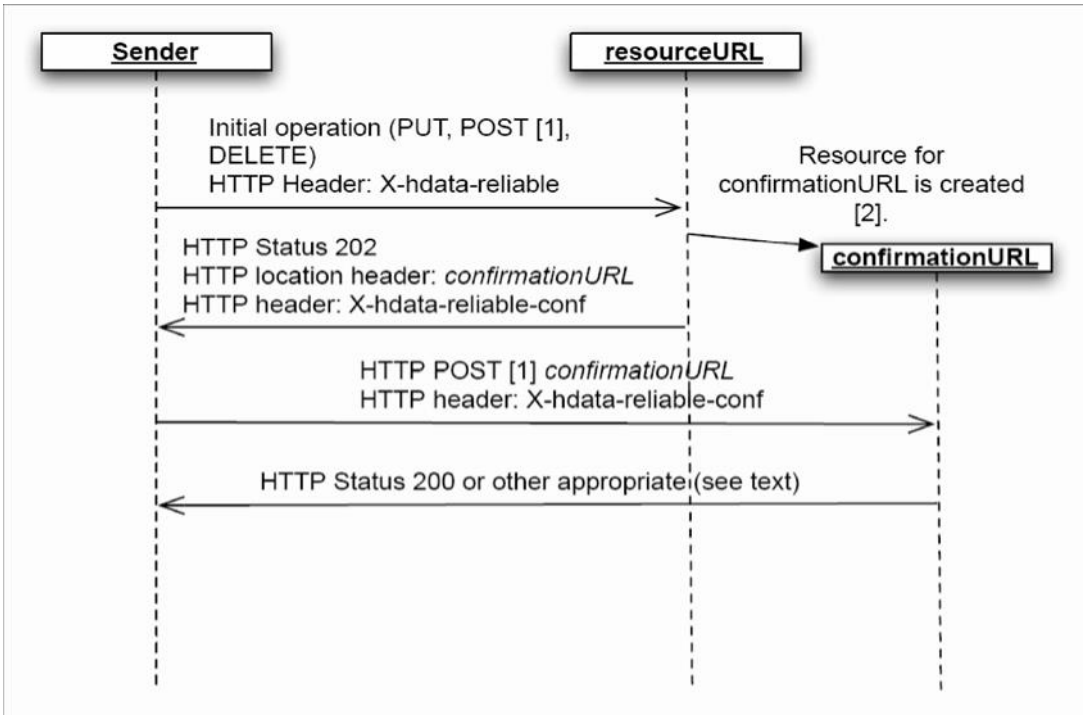
This pattern is a complex multi-step exchange, applicable to situations where reliable transfer of information is required. In the context of this specification, "reliable" means that both sender and service provider have confirmation that the other side has successfully received the information exactly once. For example, in a medication administration scenario, the sender would be the prescribing provider, and the service would be an order system that informs the staff to administer a given product. In this case, the system would need to make sure that:

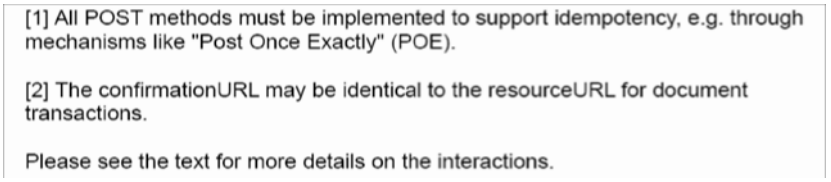
(i) the prescription is sent only once,
(ii) the sender receives an acknowledgement from the medication administration service that received the information, and
(iii) the medication service is assured that the sender received that acknowledgement.

Items (i) is guaranteed by the idempotency of the operations, item (ii) is achieved through the standard HTTP request/response pattern, and item (iii) is achieved through the pattern described in this section.

This pattern MAY be combined with any operation in section 6 when interacting with an hData Record or with other message patterns, as long as there is no overloading of HTTP methods or it is explicitly forbidden by this document or the HTTP specification.

The use of the reliable operations pattern will be governed by the business requirements of the business domain. It should be noted that this pattern breaks the statelessness of the service. As such, it cannot be used easily with load balancers and similar horizontal scaling techniques.





The flow of the patterns is as follows:

1. The sender accesses the *resource URL* resource using PUT, POST, or DELETE. To indicate that it wants to use the reliable operations pattern, it sets the HTTP message header "X-hdata-reliable."
2. If the *resource URL* is capable of performing the reliable operations pattern, it will create a new resource for a message at *confirmation URL*, and return an HTTP status code of 202. The HTTP result MUST contain the *confirmation URL* in the HTTP location header and a confirmation secret in the "X-hdata-reliable-conf" header. This secret SHOULD be a simple string of sufficient length to prevent guessing. The service MUST NOT process the message at this stage. This means that once the *confirmation URL* is created the resource is locked, until the pattern completes, or after a preconfigured time-out. The server MUST send a HTTP status code 405 to any client trying to modify that resource while the resource is locked.

   If the *resourceURL* does not implement the reliable operations pattern, it MUST return an HTTP status code of 405 and discard the message.

3. The sender MUST then POST an empty request body to the resource at *confirmation URL* and set the "X-hdata-reliable-conf" header to the value provided in step 2. Upon receipt, the service – listening at the *confirmation URL* – MUST validate the confirmation secret. Once the GET secret is validated, the service processor MUST process the message immediately.

4. If the validation is successful, the *confirmation URL* returns an HTTP result with the expected status code for the initial operation. If the validation is not successful, the service MUST return an HTTP status code of 409. The sender MUST retry the POST until it receives either a different HTTP status code.

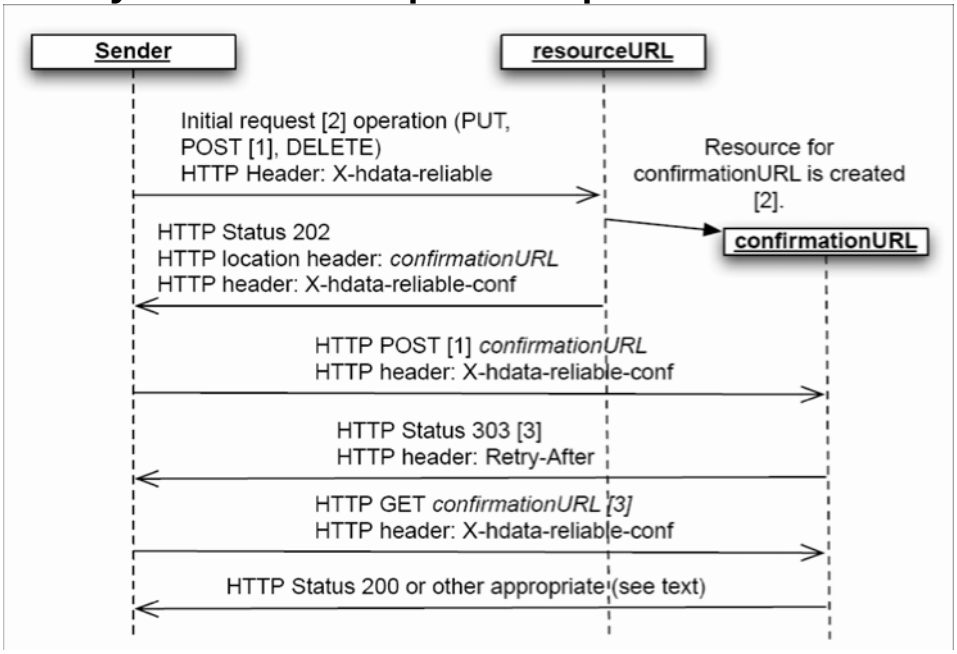
Remarks:

1. Since POST is not idempotent, the service MUST implement a safe guard against duplicity of requests for all POSTs in this flow. It is RECOMMENDED that the service implements "POST Once Exactly" (POE) [13].

2. The *confirmation URL* resource MAY be destroyed after the reliable message pattern flow is complete. The service MAY maintain the *confirmation URL* after the pattern flow completes.

3. If the initial operation in step 1 above is an application-level request message or document, the *confirmation URL* MAY provide an application-level response in step 4. The response MAY be provided by returning the response body in the final step; the HTTP status code MUST NOT be 409. For asynchronous responses, the *confirmation URL* MAY return an HTTP status 303 with a "Retry-After" header, indicating when the response will be available through a GET operation at the *confirmation URL*.

7.2 Asynchronous Request/Response Pattern



This pattern extends the Reliable Operations Pattern to enable a simple asynchronous request response pattern. It allows a service to direct a client to return at a later time and pickup the result of a given request, by using the HTTP Retry-After header.

[1] All POST methods must be implemented to support idempotency, e.g. through mechanisms like "Post Once Exactly" (POE).

[2] The request/response protocol is defined at the application level and not through this specification. The Sender and the service at the resourceURL will determine if the operation is a request.

[3] The 303/Retry-After step is optional. It MAY be used for asynchronous responses.

Please see the text for more details on the interactions.

This specification does not provide guidance to what constitutes an application-level request/response protocol. Implementers of this specification can decide if this mechanism is appropriate for their application.

1. There is no default for how long the *confirmation URL* resource is *available for* confirmation (step 3). The service MAY destroy the *confirmation URL* resource and discard the message if the sender does not complete step 3 of the pattern flow. It is strongly RECOMMENDED to advertise the maximum time for confirming the message to the developer of the sender in the documentation for the service. If the service discards the message after timing out *the confirmati*on step, it MUST return a status code of 404 at the *confirmation URL* permanently. If the service issued a "Retry-After" header in response (as indicated in Remark 3.), it MUST provide the *confirmation URL* until after the expiration of the time indicated by this header.

2. For operations on hData Records (as described in section 6) special provision MUST be taken to prevent alteration of the resource once the reliable message pattern is initiated. This means that once the *confirmation URL* is created the resource is locked, until the pattern completes, or after a preconfigured time-out. The server MUST send a HTTP status code 405 to any client trying to modify that resource while the resource is locked. The service MUST provide the old status of the resource until step 3 completes. It is RECOMMENDED to use the resource URL (which is different from the URL for the metadata for the resource URL) also as the *confirmation URL*

8 Security Considerations

This transport and API specification can be used to transfer data in many different situations, for example, inside organizations, between organizations, or from medical devices. As such, the specification cannot provide a comprehensive security solution that addresses the needs of all possible applications. However, this section describes a number of basic security mechanisms that hData implementations MUST support. In addition, this section describes general web security considerations and how additional security mechanisms and systems can be added to implementations of this standard. Implementers of hData are advised to review their domain specific security requirements and select or create appropriate security mechanisms. The section concludes with a discussion of risk analysis, which is highly recommended prior to implementing and deploying any infrastructure for clinical systems.

While this specification does not define any access controls to the web resources, it is RECOMMENDED that a comprehensive access control management system is always deployed with any hData installation.

8.1 Security Mechanism Specification

To allow the support of multiple security mechanisms at a single HRF resource, clients MUST be able to always access the (*baseURL*) through an HTTP OPTIONS request (see [8], section 9.2), as well as the (*baseURL*)/metadata resource. If the resource employs any security mechanism with the exception of transport security (see 8.2.1), it MUST include one or more the HTTP WWW-Authenticate header that MUST contain the applicable credentials that identify the supported security mechanism and the applicable authentication parameters. Section 8.2 includes the details for the baseline security mechanisms.

It is RECOMMENDED that hData Content Profiles include a detailed specification of any required custom security mechanisms. The URIs for identifying these additional security mechanisms SHOULD be made unique by using the DNS domain name in the first part of the URI. Below is the template for specifying such a security mechanism.

Any new security mechanism specification that is compliant with this standard needs to provide the following items. This SHOULD be done through a commonly readable text document, such as HTML. This package provides implementers with the necessary security protocol information to create the security mechanism for their system.

1. Common Name (REQUIRED) – free text, recommended to be less that 32 characters.
2. Identifier (REQUIRED) – URI or none, recommended to include the originating organization's DNS domain name for uniqueness. NOT REQUIRED for transport security (see 4.2.1). It is RECOMMENDED to use a URL that resolves into the HTML representation of the security mechanism specification.
3. Exclusiveness (REQUIRED) – free text, describes if the mechanism can be combined with other mechanism.
4. Description (REQUIRED) – free text, includes a comprehensive description of all allowed interaction patterns, parameters, and dependencies.
5. State diagram (RECOMMENDED) – UML state diagram, identifies all actors and illustrates all allowed interaction patterns. The state diagram SHOULD be encoded in the latest released version of XMI.
6. Business rules (RECOMMENDED) – free text, describes the business/domain justification and rules for this security mechanism.
7. Example (RECOMMENDED) – free text, recommended to include examples including packet content for all interaction patterns.
8. Other Content (OPTIONAL)

8.2 Baseline Security

The mechanisms described in this section MUST be supported by all implementation of this specification. While transport security is always RECOMMENDED, there can be situations where transport security is not required.

The versions of IETF standards selected within this specification are the minimal REQUIRED versions. It is RECOMMENDED to use more modern versions, as long as these newer versions are backward compatible.

8.2.1 HTTP Transport Security

Transport security is implemented within the network stack below the HTTP transport layer.

1. Common Name: HTTP Transport Security
2. Identifier: none – Not required because the identifier is encoded in the *base URL* URL through the https scheme.
3. Exclusiveness: This mechanism can be combined with all other security mechanism.
4. Description: Implementations MUST support TLS 1.1 or higher. This protocol is described in detail in IETF RFC 4345 [2]. TLS supports both anonymous clients, as well as client authentication. Implementations of this specification MUST support anonymous client, and MUST support client authentication through TLS. If TLS client authentication is supported, implementation MAY use the principal obtained from the exchange in their authentication and authorization process.

8.2.2 Message Security

1. Common Name: S/MIME Message security
2. Identifier: http://www.omg.org/hdata/2011/03/security/smime-messages
3. Exclusiveness: This mechanism can be combined with all other security mechanisms.
4. Description: Implementations MUST support S/MIME 3.2 or higher which is an IETF internet standard described in IETF RFC 5751 [4]. S/MIME requires PKI certificates for sender and receiver, and also a way for the sender to discover the public key certificate for the receiver. The sender should include its own certificate in the S/MIME message. Implementations MUST use SHA-256 and RSA for signature and encryption, respectively. To achieve confidentiality, implementations MUST use the EnvelopedData content type [10], section 2.4.3. The hData SectionDocument that becomes the MIME payload of the S/MIME message MUST be prepared by the implementation according to the requirements of the S/MIME specifications, with special consideration for the MIME content type.

While out of scope for this specification, there are a number of ways to discover the certificates:

- If the receiver offers any web resources through https, it is RECOMMENDED to use the server certificate.
- If any discovery services are available, it is RECOMMENDED that the metadata for the endpoint includes the public key certificate.
- If DNS CERT resource records (IETF 4398 [5]) are available, the sender MAY use the certificate published.

8.2.3 Authentication

Authentication can be achieved through all of the mechanisms described in this section. Implementations of this specification MUST support all described authentication mechanisms, but these mechanisms MAY be disabled at deploy or runtime.

8.2.3.1 HTTP Basic Authentication

1. Common Name: HTTP Basic Authentication
2. Identifier: http://www.omg.org/hdata/2011/03/security/http-basic-auth
3. Exclusiveness: This mechanism can be combined with all other security mechanisms. When combining with other authentication mechanisms, it SHOULD use the other mechanism's security principal for authentication and authorization.
4. Description: Implementations MUST implement HTTP Basic Authentication as specified in IETF RFC 2617 [6], section 2.

8.2.3.2 HTTP TLS Authentication

1. Common Name: HTTP over TLS
2. Identifier: http://www.omg.org/hdata/2011/03/security/http-tls-auth
3. Exclusiveness: This mechanism SHOULD NOT be combined with other authentication security mechanisms. If combined with other security mechanisms, the principal of the client certificate, as identified by the Common Name (CN) attribute of the certificate, SHOULD be used as the security principal in all subsequent authentication and authorization decisions.
4. Description: Implementations MUST implement HTTP TLS Client Certificates as specified in IETF RFC 2246 [7], section 7.4.6.

8.3 Specifying A Custom Security Mechanism

Additional security mechanisms that can be published through the X-hdata-security header can be created as needed by the behavioral model and the application domain. It is RECOMMENDED to include or reference security mechanisms necessary for a given hData Content Profile (HCP) within the HCP package. The security mechanism description MUST comply with the template specified in Section 8.1.

8.4 General Web Security Considerations (Non-Normative)

Because hData is implemented using common web technology, it is subject to the same security considerations as other security-sensitive web applications and services. Because Internet threats and vulnerabilities are constantly evolving, hData implementations should apply current best practices to assure appropriate levels of security.

These security best practices should be considered not only at the software application layer, but also at lower layers such as the network layer and physical layer. For example, hData implementations MAY also support lower-level protection mechanisms, such as IPSEC or other bulk traffic encryption. Typically, such technologies have no direct impact on the application layer, and their use and implementation is determined by the networking infrastructure. Protection of critical infrastructure services such as DNS or DHCP MAY be necessary. Information security must be integrated with non-IT security as well:

- Any information processing systems must be protected from intentional and unintentional physical harm, both man-made as well as natural.
- Business processes and non-IT workflow must integrate with information security, and prevent circumvention of information security measures.
- System operators and end users must be cleared for access at the appropriate level.

The reader is advised to consult appropriate resources in this area for more information, such as NIST 800-12, NIST 800-14, ISA-99, and ISO 27002.

8.5 Risk Assessment Approach and Best Practices (Non-Normative)

It is highly RECOMMENDED to perform a comprehensive risk analysis prior to deploying any clinical application. Risk analysis is a systematic consideration of the threats, vulnerabilities, and consequences of gaps in security, as well as mitigation strategies for risks. Often, the threats and vulnerabilities are captured in terms of specific scenarios that can be re-used during security audits throughout the system's lifecycle. The reader is advised to consult appropriate resources for more information on cyber risk assessment, such as NIST SP 800-30, SP 800-37, SP 800-39, the IHE security cookbook [11], and ISO/TS 25238.

9Realization of RLUS Profiles (Non-Normative)

9.1 Introduction

The Retrieve, Locate, Update Service (RLUS) Specification defines an HL7 framework for healthcare services. The hData RESTful Transport is a realization of RLUS Functional Profiles. The hData Content Profile (HCP) [1], section 3, acts as such as a Semantic Profile in the sense of [5], section 6.1. Taken together, the two portions of the hData specification forms an RLUS Conformance profile. This section provides a mapping between the hData RESTful implementation and the RLUS framework.

It should be noted that while this section is necessary to establish hData as a Platform Specific Module of the OMG RLUS Platform Independent Module, it does not require any additional implementation burden on the developer.

9.2 Implementation of RLUS Interfaces

The RLUS specification defines a number of interfaces in [9], Section 5.4 "Detailed Functional Model." These are mostly implemented by the hData specification, as detailed within the table below. Note that a SectionDocument is the hData realization of a RLUS Resource.

Table 9.1 - RLUS Runtime/Management and Query Interface

HL7 RLUS SFM (CIM) – RLUS Basic Runtime Capabilities	**OMG RLUS STM PIM Management and Query Interface ([13])**	**hData RESTful Platform Specific Model (PSM) Implementation**	**Note**
Locate Resources (4.4.1)	Locate (7.4)	GET (baseURL) GET (baseURL/sectionpath)	Parameter-specific query may be implemented either over a single HDR or a collection of HDR by another specification. This is out-of-scope for the HRF and this specification.
Get Resource (4.4.2)	Get (7.2)	GET (baseURL/sectionpath/ documentname)	This is implemented using an HTTP GET operation on the resource identified by its URL.
List and Get Resource (4.4.3)	List (7.3)	Not implemented	The Atom 1.0 feed returned at each Section level as well as at the *baseURL* (see Sections 6.4.1 and 6.2.1, respectively) implements the List Interface
Put Resource (4.4.4)	Put (7.5)	POST (baseURL/sectionpath)	Sub clause 6.4.2.2 (Add new document) describes how a new SectionDocument can be created.

Table 9.1 - RLUS Runtime/Management and Query Interface

Initialize Resource (4.4.5)	Initialize (7.8)	Not implemented	The initialization of a resource and the actual creation is always performed in a single transaction within hData. As such, when creating a new SectionDocument as described in Section 6.4.2.2, hData returns the location of the newly created resource as part of the transaction. As such, this operation by itself makes no sense in the hData RESTful context
Discard Resource (4.4.6)	Discard (7.6)	DELETE (baseURL/sectionpath)	Section 6.5.4 (DELETE) describes how a SectionDocument can be deleted.

Section 5.6 in the HL7 RLUS SFM describes the Introspective Capabilities, which are mapped to hData in the following table.

Table 9.2 - RLUS Introspective/Semantic Profiles Interface

HL7 RLUS SFM (CIM) – Introspective Capabilities	**OMG RLUS STM PIM Semantic Profiles interface (version 1.0.1, formal/2011-07-02)**	**hData RESTful Platform Specific Model Implementation**	**Note**
List Conformance Profiles (4.6.1)	List Conformance profiles (13.6)	OPTIONS (baseURL)	Section 6.2.5 (OPTIONS) describes the X-hdata-hcp header which returns a list of hData content profiles.
List Semantic Signifiers (4.6.2)	List Semantic Signifier (13.5)	GET (hDataRoot/root.xml) or OPTIONS (baseURL)	The root document at *baseUrl*/root contains of supported elements within the Extensions node. The list of Extension elements represents the list of semantic signifiers, as required by [5] 5.2.1. (The HRF specification [1] recommends URLs as identifiers for each Extension, which should resolve into a RDDL document describing the given Extension. This is consistent with the recommendation of [5] section 5.2.1 to provide an explanation for each semantic signifier.) Alternatively, the list of Extension can also be obtained throught the OPTIONS request against the baseURL and the evaluation of the X-hdata-extension HTTP header (see section 6.2.5).

Table 9.2 - RLUS Introspective/Semantic Profiles Interface

Describe Semantic Signifier (4.6.3)	Describe (7.7)[a] Find Semantic Signifier (13.3)	GET (url)	For any <Extension> that is a URL and resolves into a RDDL document, the necessary description can be retrieved. Thus, if an hData implementation strives to be compliant to this interface, recommendation in [1] section 2.3 to use URLs and resolve into RDDLs becomes a requirement.
Put Semantic Signifier (4.6.4)	Create Semantic Signifier (13.2) Update Semantic Signifier (13.4)	Not implemented	hData does not allow explicit creation of new Extensions for a given system. However, if the system supports Extensions that are not currently registered in the root document, they can be added to the record by creating a new Section as described in Section 6.2.2 and 6.4.2.

B. For pragmatic reason Describe operation, currently, is included in the Management and Query Interface of the OMG. PIM

Since the above mapping provides the Basic Runtime and the Introspective Capabilities, hData implements RLUS at Level 2 (see [9], section 6.2).

Annex A – Normative References

[1] G. Beuchelt et al., "hData Record Format V1", Draft Standard for Trial Use, Health Layer 7, 2012, online at http://www.hl7.org/documentcenter/public/standards/dstu/HL7_V3_ITS_HDATA_RF_DSTU_R1.pdf

[2] T. Dierks, E. Rescorla "Transport Layer Security (TLS) 1.1," RFC 4346, IETF, April 2006, online at http://tools.ietf.org/html/rfc4346

[3] M. Nottingham, R. Sayre, "The Atom Syndication Format", RFC 4287, IETF, December 2005, online at http://tools.ietf.org/html/rfc4287

[4] B. Ramsdell, S. Turner, "S/MIME 3.2 Message Specification," RFC 5751, IETF, January 2010, online at http://tools.ietf.org/html/rfc5751

[5] S. Josefsson, "Storing Certificates in the Domain Name System (DNS)" RFC 4398, IETF, March 2006, online at http:// tools.ietf.org/html/rfc4398

[6] J. Franks, et al., "HTTP Authentication: Basic and Digest Access Authentication," RFC 2617, IETF, June 1999, online at http://tools.ietf.org/html/rfc2617

[7] T. Dierks, C. Allen, "The TLS Protocol Version 1.0," RFC 2246, IETF, January 1999, online at http://tools.ietf.org/html/rfc2246

[8] R. Fielding, et al., "Hypertext Transfer Protocol – HTTP 1.1," RFC 2616, IETF, June 1999, online at http://tools.ietf.org/html/rfc2616

[9] HL7 Resource Location and Updating Service (RLUS), DSTU Release 1, Health Level Seven, Inc., December 2006, online at http://www.hl7.org/documentcenter/public/standards/dstu/2006SEP/SUPPORT_POOL_V3_RLUS_R1_D1_2006SEP_20061220112743.pdf

[10] B. Ramsdell, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification," RFC 3851, IETF, July 2004, online at http://tools.ietf.org/html/rfc3851

[11] "Cookbook:Preparing the IHE Profile Security Section," IHE International, October 2008, online at http://www.ihe.net/Technical_Framework/upload/IHE_ITI_Whitepaper_Security_Cookbook_2008- 11-1 0.pdf

[12] J. Snell, "The Atom 'deleted-entry' Element', RFC 6721, IETF, September 2012, online at http://tools.ietf.org/html/rfc6721

[13] "Retrieve, Locate, and Update Service (RLUS) Specification Version 1.0.1", Object Management Group, July 2011, online at http://www.omg.org/spec/RLUS/1.0.1

[14] "Unified Modeling Language Version 2.4.1", Object Management Group, August 2011, online at

http://www.omg.org/spec/UML/2.4.1

[15] "MOF 2 XMI Mapping Version 2.4.1", Object Management Group, August 2011, online at http://www.omg.org/spec/XMI/2.4.1

Annex B - Non Normative POST Example

The following example illustrates the wire-level representation of an HTTP POST operation adding a new SectionDocument (see also Section 2.4.2.2) using a simplified payload.

```
POST /example. com/additionalPatientInfo/patient12 34/allergies/ HTTP/1 .0
Content-Length: 1105
Content-Type: multipart/form-data; boundary=END _OF _PART
- - END _OF _PART
Content-Disposition: form-data; name="content"
Content-Type: application/xml

 <allergy:allergy xmlns:allergy="http://projecthdata.org/hdata/schemas/2009/06/
allergy">
   <allergy:product codeSystem="2.16.840.1.113883.6.88" code="310965" />
   <allergy:narrative>Ibuprofen allergy</allergy:narrative>
</allergy: allergy>
- - END _OF _PART
Content-Disposition: form-data; name="metadata"
Content-Type: application/xml

<hrf-md : DocumentMetaData>
   <hrf-md:DocumentId>allergy1 .xml</hrf-md:DocumentId>
   <hrf-md : RecordDate>
       <hrf-md: CreatedDateTime>
           2009-10-10T09 :21: 55Z
       </hrf-md: CreatedDateTime>
       <hrf-md:Modified>
           <hrf-md :ModifiedDateTime>
               2011-08-13T18 :30: 02Z
           </hrf-md:ModifiedDateTime>
```

```
        </hrf-md: Modified>
    </hrf -md : RecordDate>
    <hrf-md: LinkedDocuments>
        <hrf-md: LinkInfo>
            <hrf-md: Target>
                http://example.com/additionalPatientInfo/patient1234/allergies
            </hrf-md : Target>
                                </hrf-md: LinkInfo>
                            </hrf -md: LinkedDocument s>
</hrf-md : DocumentMetaData>
- -END _OF _PART- -
```

Annex C – Non Normative JSON Encoding Rules

The rules in this annex are intended to help mapping XML-based structures to a JavaScript Object Notation (JSON) format. The JSON format for hData Section Documents should follow the standard XML format closely so XPath queries can easily be mapped to query the JSON structures. As such, the following design choices should be considered:

- The names for the JSON object members are the same as the names of the elements and attributes in XML, including for elements that may repeat.
- In the data types, the value representations for primitive types such as string, decimal, etc. are identical between the XML and the JSON form (including instant, which is represented as plain text, not in one of the proposed JSON custom date formats)
- Just as in XML, JSON property attributes never have empty values; omit a value if it is empty.

There are differences too:

- There are no namespaces in the JSON representation.
- JSON does not have a notion of attributes versus elements, so FHIR's only attributes (dataAbsentReason and id) are treated as JSON object members (see below for more details).
- JSON has a notation for arrays, which are used to represent repeating elements. Note that this is the case, even if they do not repeat in the actual instance.
- The XHTML element in the Narrative datatype is represented as a single escaped string of XHTML

These differences - particularly the repeating element one, which cannot be avoided - ensure that generic XML to JSON converters are not able to perform correctly. The guidelines below will provide a constrained, but 1-on-1 mapping between a JSON and an XML representation.

JSON representation of primitive types
XML elements with primitive values are represented as JSON object members of the same name, with their value encoded as a string. Native JSON types other than "string" are never used, to guarantee equivalence of the serialized representation between XML and JSON. Primitive elements used inside the datatypes cannot have an 'id' or 'dataAbsentReason' attribute, so they are rendered in JSON as a property and value:

```
<date>"1972-11-30"</date>
```

is represented in JSON as

```
"date": "1972-11-30"
```

Primitive elements inside resources *can* have 'id' or (if allowed) 'dataAbsentReason' attributes, so their JSON representation uses a JSON object with members '_id', 'dataAbsentReason' and 'value', which contains the actual primitive value as string. Note that in most cases, the primitive will not have a value when there is a dataAbsentReason present, in which case the special 'value' member must not be present. So,

```
 <dob id='314159'>1972-11-30</dob>
```

is represented in JSON as:

```
"dob": { "_id": "314159", "value": "1972-11-30" }
```

while this example would look like this in the case of a dataAbsentReason:

```
"dob": { "_id": "314159", "dataAbsentReason": "Unknown" }
```

Repeating elements
Repeating elements are rendered withing a JSON array with the name of the element, so a repeating <dob> element in

```
 <dob>2011-11-30</dob>
 <dob id='314159'>1972-11-30</dob>
```

is represented in JSON like so:

```
"dob": [
     { "value": "2011-11-30" },
     { "_id": "314159", "value": "1972-11-30" }   ]
```

JSON representation of composite datatypes
Resources and other composite datatypes (types that contain named elements of other types) are represented using a JSON object, containing a member for each element in the datatype. Composites can have id's and dataAbsentReasons, therefore these attributes get converted to JSON members values, in the same manner as described for primitives. These members will be placed before all other members. For example:

```
<Person>
     <id>34234</id>
     <name>
          <use>official</use>
          <part>
               <type>given</type>
               <value>Karen</value>
          </part>
          <part id="n1">
```

```
            <type>family</type>
            <value>Van</value>
        </part>
    </name>
    <text>
        <status>generated</status>
        <div xmlns="http://www.w3.org/1999/xhtml">...</div>
    </text>
</Person>
```

is represented in JSON as:

```
{   "Person" : {
    "id" : { "value" : "34234" },
    "name" : [
        {
            "use" : "official",
            "part" : [
                {
                    "type" : "given",
                    "value" : "Karen"          },
                {
                    "_id" : "n1",
                    "type" : "family",
                    "value" : "van"
                }
            ]
        }
    ],
    "text" : {
        "status" : "generated",
        "value" : "<div xmlns='http://www.w3.org/1999/xhtml'>
                    ...</div>"
    }
}
```

Things to note here are:

- Because the primitive element 'id' is in a resource, it is serialized as a JSON object.
- In the family part of 'name', the '_id' is added as the first member.
- The 'code' within member 'language' is a part of the resource definition and therefore serialized as a JSON object, whereas the member 'use' within 'name' is a primitive inside a datatype and therefore serialized as a JSON string value.
- The XHTML content in the 'div' element which is in the Narrative element 'text' is represented as an escaped string in the value property in JSON. The xhtml's root element needs to be a in the xhtml namespace.

JSON Bundles (Atom feeds)

There are several initiatives to define a JSON format for Atom, all of which are suffering from having to simulate properties, namespaces and other xml-specific features. Instead, FHIR defines a JSON format that is tailored to the specific needs for bundles. Each element in the Xml feed definition has a JSON corresponding JSON object member with the same name. Here is an example feed returning search results for a person query:

```
{
     "title" : "Search result",
     "updated" : "2012-09-20T12:04:45.6787909+00:00",
     "id" : "50ea3e5e-b6a7-4f55-956c-caef491bbc08",
     "link" : [
          {
               "rel" : "self",
               "href" : "http://ip-
                         0a7a5abe:16287/fhir/person?format=json"
          }
     ],
     "entry" : [
          {
               "title" : "Resource of type Person,
                           with id = 1 and version = 1",
               "link" : [
                    {
                         "rel" : "self",
                         "href" : "http://example.com/
                                   person/@1/history/@1"
                    }
               ],
               "id" : "http://example.com/
                         person/@1",
               "updated" : "2012-05-29T23:45:32+00:00",
               "published" : "2012-09-
                              20T12:04:47.3012429+00:00",
               "author" : [
                    {
                         "name" : "John Doe"
                    }
               ],
               "category" : [
                    {
                         "term" : "Person",
                         "scheme" : "http://example.com/
                              /resource-types"
```

```
                    }
               ],
               "content" : {
                    "Person" : { ...person content in JSON... }
               },
               "summary" : "<div
               xmlns=\"http://www.w3.org/1999/xhtml\">(text
               summary)</div>",
               "version" : "1"
          },
          ... other entries ....
     ]
}
```

Note that property names for elements that can repeat are not pluralized for consistency of overall approach.

Bundling versions - deletion
When returning a set of versions for a resource, a version might indicate a *deletion*. While the XML format follows RFC 6721, the JSON format needs to use an entry item to retain the logical order of entries:

```
   .. feed ..
"entry" : [
     ... other entries ....,
     {
          "deleted" : "2012-05-29T23:45:32+00:00",
          "link" : [
               {
                    "rel" : "self",
                    "href" : "http://example.com/
                              person/@1/history/@1"
               }
          ],
          "id" : "http://example.com/person/@1",
     },
     ... other entries ....
]
... feed ...
```

The entry is known to be deleted because a date of deletion is given. An id must be provided, and a link may be provided.

Binary Resources
There are situations where it is useful or required to represent binary resources in the bundle.

These are resources that are referred to from Document (usually via a URI), and where it is convenient to include these in the feed directly rather than leaving them by reference. When binary resources are represented in the feed, they are enclosed as base64 encoded content along with a content-type, which is the mime-type as it would be specified in HTTP:

```
  ...
"content" : {
     "Binary" : {
          "contentType" : "[mime type]",
          "content" : "[base64 of data]"          }        },
     "summary" : "<div
     xmlns=\"http://www.w3.org/1999/xhtml\">Binary
     Resource</div>"   ...
```

The summary should be some equivalent to "Binary Resource" in an appropriate language.